Deep Learning

Motive :- Basics, Maths, Interview Preparation

Day-1

Agenda

1) Deep learning → Perception {AI vs ML vs DL vs DS}
2) Forward propagation
3) Backward Propagation.
4) Loss function
5) Activation functions
6) Optimizers

Prerequisite

① Python
② ML
③ Stats.

Q. Why deep learning is becoming populare?
→ Good Model Creation.
→ Hardware Advancement.

DL minimizes the need for human action since its algorithms conduct feature extraction on their own. This makes the process much faster and reduces the risk of human error.

suppose
$y = 1$ (actual value)
$\hat{y} = 0$ (predicted value)

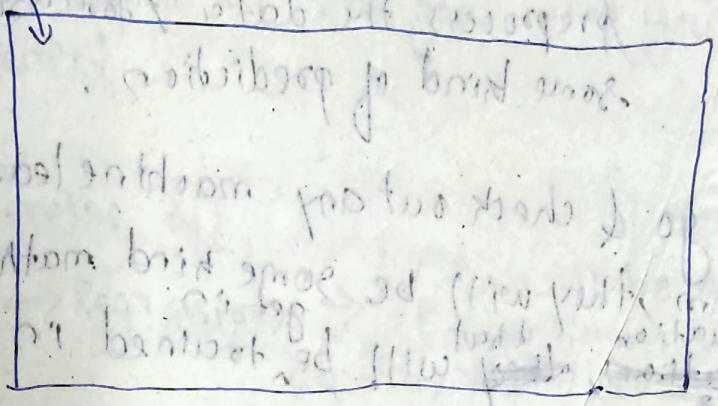
then $y - \hat{y} = 1$ (this diff should be very near to 0)

→ loss function

AI vs ML vs DL vs DS

AI → Artificial Intelligence
ML → Machine Learning
DL → Deep Learning
DS → Data Science

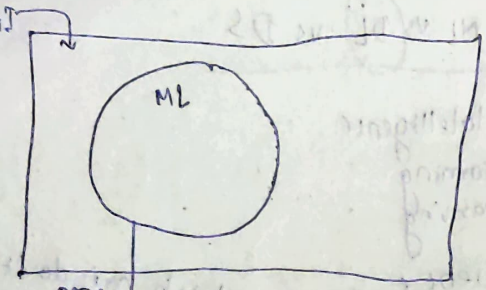
AI (It's an application which can do its own task without any human intervention)



At the end of the day in creating an AI application.

Ex:- Netflix → Action → Recomendation
Self driving cars → Comedy → "
Youtube
Amazon.in

AI → It means you have created an application and that application is performed without any human interference/intervision.



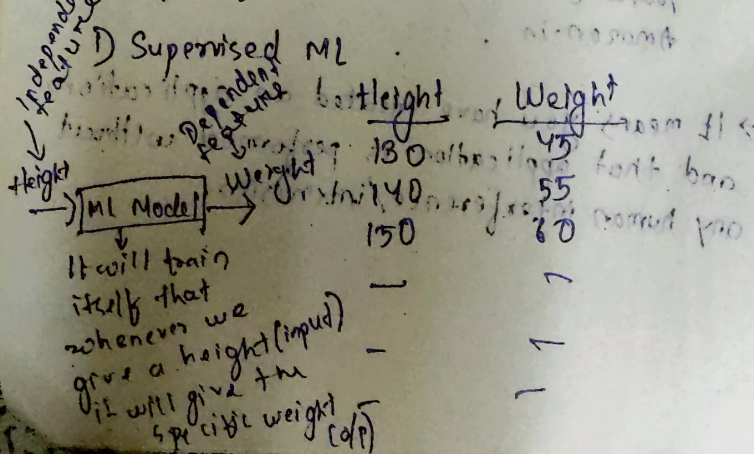
↳ Stats tools to analyse the data / visualize the data / preprocess the data / forecasting some kind of prediction.

If you go & check out any machine learning algorithm, they will be some kind mathematical equation that will be trained in the data.

These mathematical equation can considered as tools.
stats

Different types of Machine Learning techniques :-

1) Supervised ML



In this ML Model :

if we give height it will be the independent feature and weight as the dependent feature (output feature)

In short

→ In a supervise ML problem statement you will definately have an output feature. So, you know us end goal what you need to predict.

1) Supervise ML

→ Regression → o/p → will be a continous value
Eg:- 45, 46, 47, 46·5, 46·3, ---

→ Classification → o/p → fixed no. of categories

Eg:-

Independent Height	Independ Weight	dependent Gender
—	—	M
—	—	F

→ Binary classification

In Gender classification, the above type will be binary classification as it only have two possible outputs.

If there will be more than one categories then two outputs then it will be Multiclass classification.

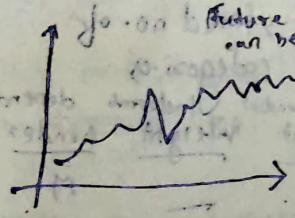
3) Multilabel Classification:-

(This multilabel classification is used in recomendation system as such)

Movies	~~cat1~~ Action	~~cat2~~ Romance	~~cat~~ Thriller	Suspense
Endgame	1	0	1	1
~~Rudra~~ Dr Shyam	0	0	1	1

With respect to movies it can based on various categories, so based on ~~pro~~ probability we usually give us o/p.

forecasting? → Regression problem.



→ We usually this type of problem statement in Time series analysis.

Multiclass classification	Multilabel classification.
→ In multi class, it has just one output only, and many no. of categories	→ In multilabel, it has many categories as well as many outputs.

2) Unsupervised ML :-

(clustering, segmentation, reduce dimension)

Eg:-

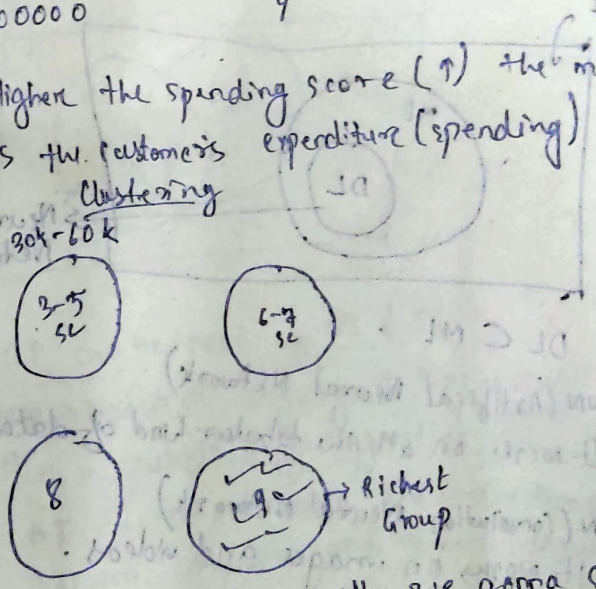
Company → product → costly product

Company has some kind of customer data such as

Salary	Spending score (sc) (0-9)	Email → product
65000	3	
70000	5	
90000	4	
100000	9	

Higher the spending score (↑) the more is the customer's expenditure (spending)

Clustering



If the product is costly we gonna send the mail to only the richest Grp

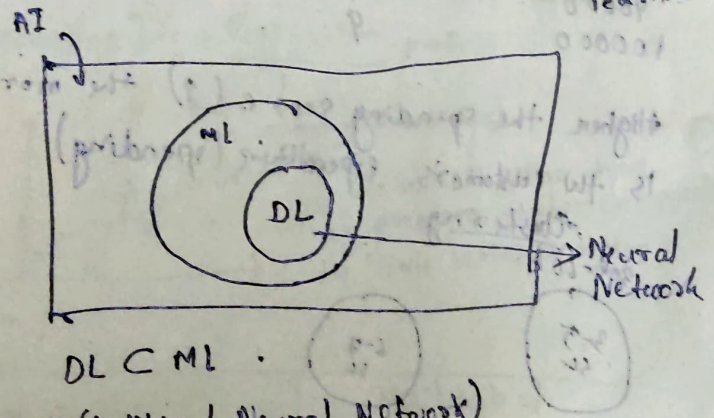
3) Semi supervised ML :- (Reinforcement)

In this ML, we gonna have :-

Labeled data → Model → Recomendation

(Ex- Netflix) with the interaction with Mobile the films u watched such as action, romance → Model → New Recomendation.

Deep Learning → Mimic human Brain → Make the machine learn.



DL ⊂ ML .

1) ANN (Artifical Neural Network)
It works on all the tabular kind of data.

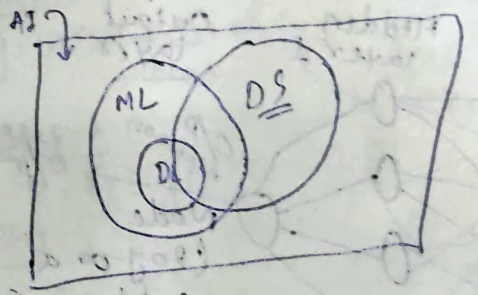
2) CNN (Convolution Neural Network)
It works on images and videos.

Applications :- 1) Image classification
2) Object detection → RCNN, FAST RCNN, FASTER RCNN, SSD, YOLO
3) Object Segmentation
4) Tracking
5) GAN

3) RNN (Recurrent Neural Network)
It works on text, time series and sequential data

Techniques in RNN
→ RNN
→ LSTM RNN
→ Bidirectional LSTM RNN
→ Encoder, Decoder
→ Transformers
→ BERT
→ GPT1, GPT2, GPT3

Data Science



NLP can be part of ML & DL.
NLP → Natural language Processing
It is a subfield of linguistics, computer science and AI concerned with the interactions between computers and human language.

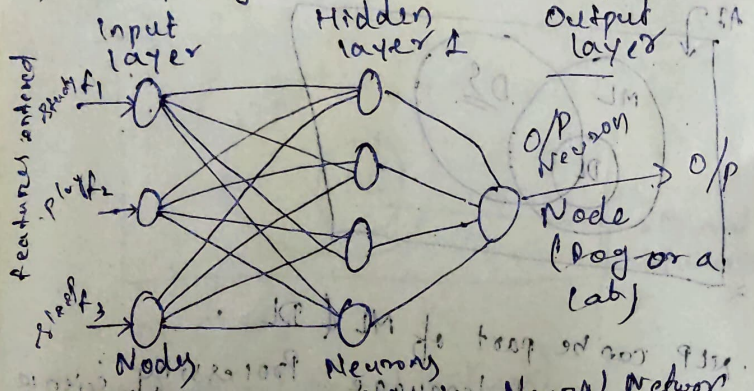
Deep Learning

1) ANN → (Artifical Neural Network)
2) CNN → (~~Comp~~ Convolutional Netural Network)
3) RNN → (Recurrent Neural Network)

It is a technique to mimic of human brain.

Neural Network

→ Perceptron {single layered Neural N/W}
→ Back propagation (Jeffren Hinton)



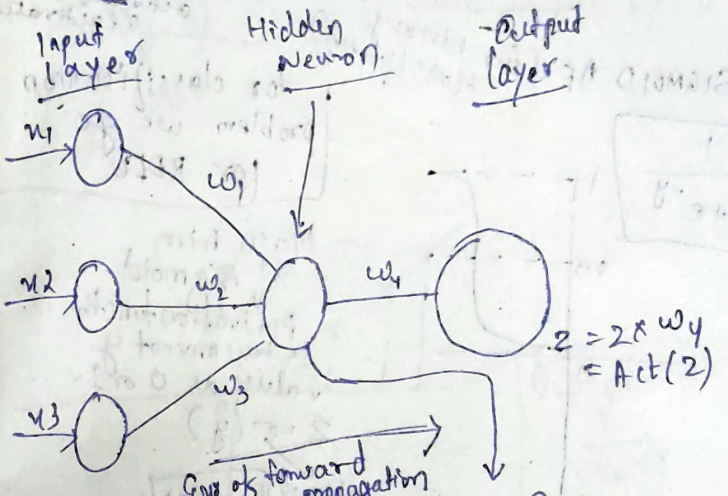
This is a single Neural Network architecture

ANN

Neuron → does some kind of signal processing.

Dataset	Study	Play	Sleep	Pass/fail
→	7	3	7	1
→	2	5	8	0
→	4	3	7	1

How Neural Network Works



Step 1

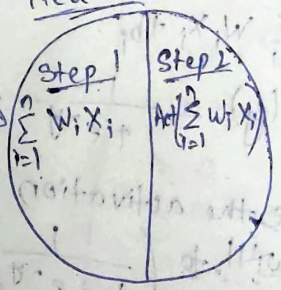
$y = w_1x_1 + w_2x_2 + w_3x_3 + \text{bias}$

Step-2

$z = Act(y)$

Step →

$z = z * w_4 = Act(z)$



$w_1x_1 + w_2x_2 + w_3x_3$

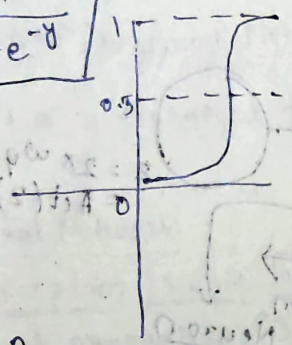
Bias → By mistake if you have initialized all your weights as zero bias will have some value so that we can continue the training.

Bias will be required in every hidden layer, there must be 1 bias.

Activations Functions (whether the neurons should be activated or deactivated)

1) SIGMOID AF (for binary classification)

$\frac{1}{1+e^{-y}}$



for classification problem we go for RELU

Main Aim of sigmoid Activation function is to convert y value as 0 or 1.

$z = \sigma(y)$

sigmoid

b_i : bias

$y = \sum_{i=1}^{n} w_i x_i + b_i$

for AF → we solve non linear problems.

$Act(y) = \frac{1}{1+e^{-y}}$

Once the activation function is transferred it will be $\left(\frac{1}{1+e^{-y}}\right)$ then the value ~~or~~ will be between 0 & 1.

If the value is less than 0.5, it will be considered as 0 and

If the value is more than 0.5, it will be considered as 1.

0.5 = threshold.

o/p of sigmoid AF = 0 or 1

0.5 ~~0~~ neuron isn't activated (less than 0.5)

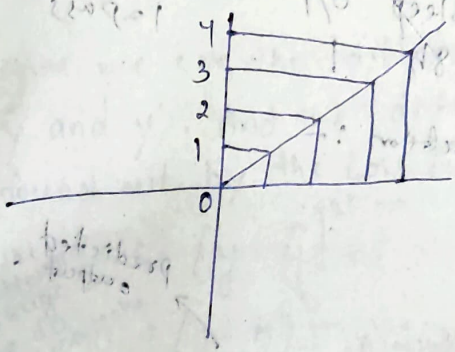
0.5 ~~1~~ = neuron is activated. (greater than 0.5)

2) RELU AF

$\max(y, 0)$

$y = \sum_{i=1}^{n} w_i x_i + b_i$

for regression problem we go for RELU



If y = -ve

$\max(-ve, 0) = 0$

If y = +ve

$\max(+ve, 0) = +ve$.

1) Sigmoid

$= \frac{1}{1+e^{-y}}$

$= \frac{1}{1+e^{-(\sum x_i w_i + b)}}$

o/p ⇒ 0 or 1

≥ 0.5 = 1

< 0.5 = 0

Importance of weights :-

Common ques :- If you put a hot object in your right hand, will you not put your hand away, bcz the neurons on the right hand will get activated (i.e. Neuron will pass signal to brain saying that a hot object is placed and you have to remove your hand) so this weight will actually help your neurons to act.

weights basically says that how much the neurons get activated or deactivated.

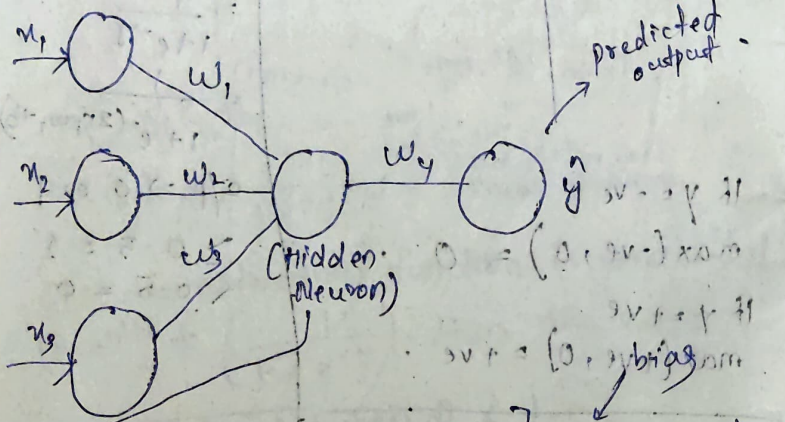
Neural Network Training with Back Propagation

we'r having a dataset.

(x_1) Play	(x_2) Study	(x_3) Sleep	O/P
2h	4h	8h	1

0 = fail
1 = pass

Forward Propagation :-



$$\begin{cases} y = [x_1 w_1 + x_2 w_2 + x_3 w_3] + b \Rightarrow \text{step-1} \\ z = Act(y) \Rightarrow \text{step-2 (The activation fu}^n \text{ may be sigmoid)} \end{cases}$$

Sigmoid transferes the value bet^n 0 – 1.

Suppose $\hat{y} = 0$ ... (predicted value) and [and we already know that $y = 1$ (output)] (real value)

and we know, we need to compare this that $\hat{y}$ and y are almost same.

→ Here we see the differences between $\hat{y}$ and y. And the differences can be found out by the Loss fu^n.

Loss fu^n = diff. between real & predicted value.

If the difference is huge we do back propagation.

$$\text{Loss} = (y - \hat{y})^2$$

$$= (1 - 0)^2$$

$$= 1$$

AIM :- To minimize the difference so that $\hat{y} \approx y$.
If the diff is ↑ we've to update the weights which can be done through BP.

⇒ As we see the loss value is higher i.e 1 and by the difference of y & $\hat{y}$ it is completely & wrongly predicted

→ So, in order to make it correct we need to make changes in w_1, w_2, w_3 to make the $\hat{y} = 1$.

→ The loss value should be reduced to a minimal value such that $\hat{y}$ should match y.

⇒ During the training phase we need to changes the w_1, w_2, w_3 & to decrease the loss values. And it can be done with the help of optimizers.

Ex of

Optimizers :- ~~Gradindescent and~~

Gradient descent

Stochastic gradient descent

and many more

Backward Propogation (Main aim of BP is to update weights)

As we got the loss value = 1 :

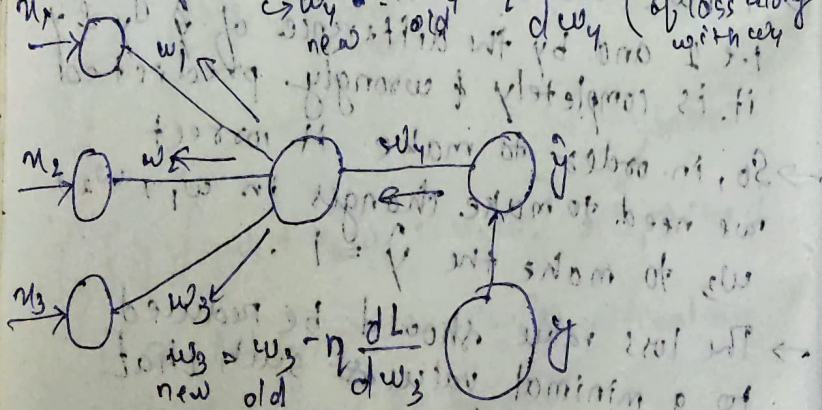
Our optimizer will try to reduce the loss value.

in order to reduce the loss value, we basically have to back propogate now.

It means we have to update the weights

η = learning rate

updated value $\rightarrow w_{4_{new}} = w_{4_{old}} - \eta \frac{dL}{dw_4}$ (derivative of loss along with w_4)



$w_{3_{new}} = w_3 - \eta \frac{dL}{dw_3}$

η = Learning rate

And Learning rate should be a very small value :

Let us consider $\eta = 0.001$

⇒ The reason of taking η into a small value because it will help us to go to the global minimal function of the gradient descent whenever we want to actually achieve the particular point.

⇒ That is only possible by an optimizer.

Suppose we're considering gradient descent.

⇒ As here, it has an single row of record over here we derived a lost function.

⇒ If we have multiple rows of record at that time we need to define the cost function

$$Cost = \sum_{i=1}^{n} (y - \hat{y})^2$$

Conclusion

FP

1) I/P layers
2) weights
3) Bias
4) Activation function

BP

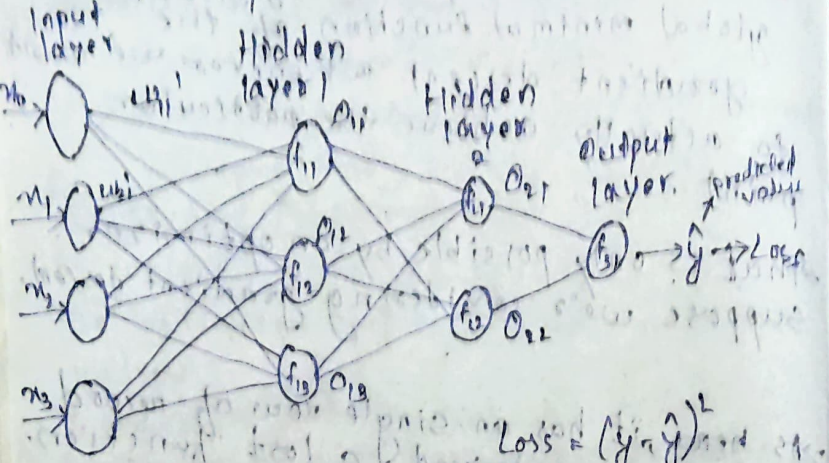
5) Loss fu? $(\hat{y} - y)$ minimize the value
6) with the help of Optimizer
7) update the weights
8) update the bias also if needed

weight updation formula :-

$$w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$$

$\frac{dL}{dw_{old}}$ ⇒ slope

Multilayere Neural Networks



Loss = $(y - \hat{y})^2$

optimizer → Loss ↓

Modified: I/p layere = 4 × 3
HL1 → 3 × 2
HL2 → 2 × 1

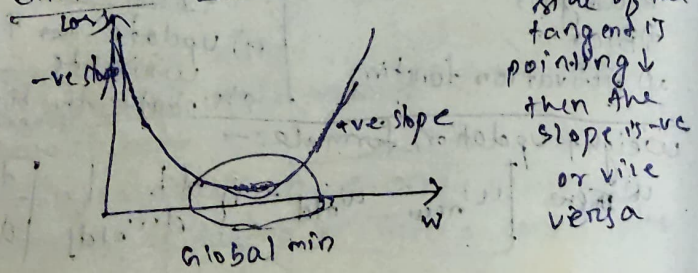
① Weight Updates

$$W_{new} = W_{old} - \eta \frac{dL}{dW_{old}}$$

Q) why we found out derivatives?

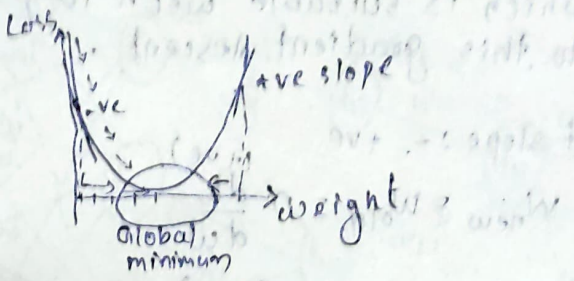
→ Derivative basically means we need to find out the slope.

GRADIENT DESCENT → kind of optimizer



Right hand side of the tangent is pointing ↓ then the slope is -ve or vice versa

Derivatives will help us to find out whether this slope is +ve or -ve!



If slope = -ve (-ve)

$W_{new} = W_{old} - \eta \frac{dL}{dW_{old}}$

$= W_{old} - (+) \cdot (-)$

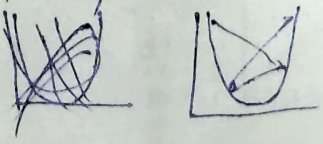
$= W_{old} + (\ )$

Adding some values with older weight means points in graph is moving down towards the global minimum.

But the speed of going down depends on learning rate (η).

If learning rate is small it will go slowly

If learning rate is very higher value then the points will jump from here and there and it may never reach the global minimal point.



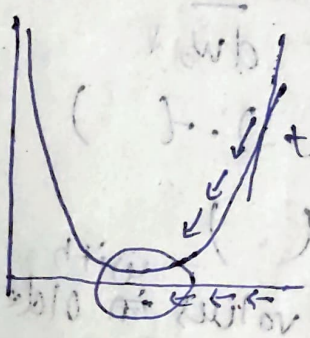
So,

It is always says that we should select a learning rate in such a way that it should be a value which is suitable with respect to this gradient descent.

If slope :- +ve (+ve)

$$W_{new} = W_{old} - \eta \frac{dL}{dw}$$

$$= W_{old} - (+ve)$$



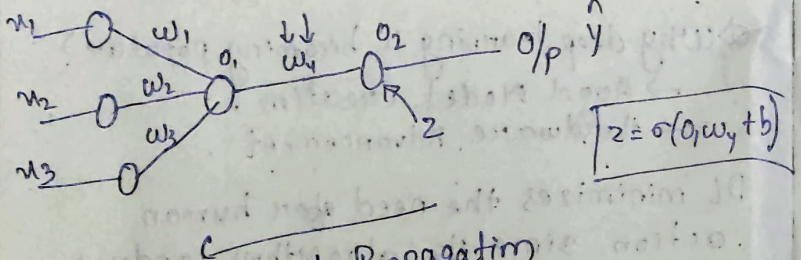
$$FP + BP = 1 \text{ Epoch}$$

Day-2

Agenda

1) forward Propagation ——→ in front side.
2) Chain Rule of Derivatives
3) Vanishing Gradient Problem
4) Loss function
5) Activation function

2) Chain Rule of Differentation L = Loss



Backward Propagation

$$w_{new} = w_{old} - \eta \left[\frac{dL}{dw_{old}}\right]$$

→ How the weight 'w4' will be updated.

tw $$w_{4new} = w_{4old} - \eta \frac{dL}{dw_{4old}}$$

→ To find this value we particularly use chain rule of derivation

$$\frac{dL}{dw_{4old}} = \frac{dL}{do_2} * \frac{do_2}{dw_{4old}}$$

updation in B.P

So, like weight ~~activation function~~, there is also bias ~~activation function~~ updation

$$b_{2new} = b_{2old} - \eta \left[\frac{dL}{db_{2old}}\right]$$

Note :-
Bias are basically the values that are added to the sums calculated at each node (except input nodes).

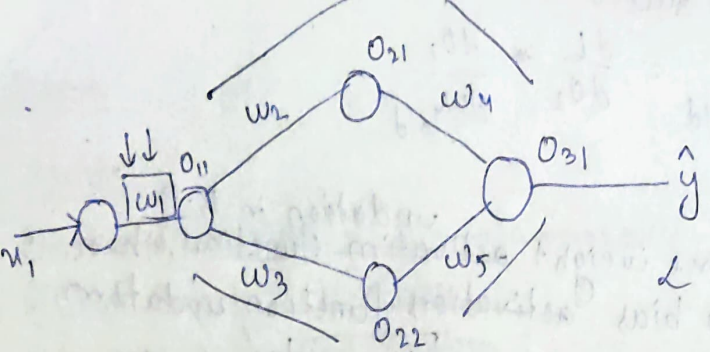
→ If we won't to ~~find~~ update the weight of w_1,

tw $$w_{1new} = w_{1old} - \eta \frac{dL}{dw_{1old}}$$

$$\frac{dL}{dw_{1old}} = \frac{dL}{do_2} * \frac{do_2}{do_1} * \frac{do_1}{dw_{1old}}$$

tw2 $$w_{2new} = w_{2old} - \eta \frac{dL}{dw_{2old}}$$

$$\left[\frac{do_2}{dw_4} * \frac{dw_4}{do_1}\right]$$

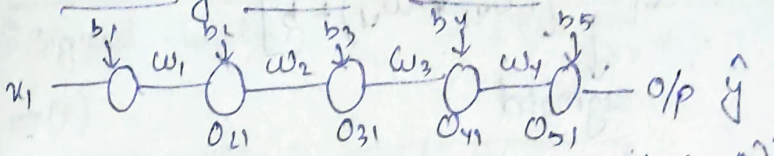


$$w_{1_{new}} = w_{1_{old}} - \eta \frac{dL}{dw_{1_{old}}}$$

$$\frac{dL}{dw_{1_{old}}} = \left[\frac{dL}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dO_{11}} * \frac{dO_{11}}{dw_{1_{old}}}\right] + \left[\frac{dL}{dO_{31}} * \frac{dO_{31}}{dO_{22}} * \frac{dO_{22}}{dO_{11}} * \frac{dO_{11}}{dw_{1_{old}}}\right]$$

There are basically 2 ways to go from L to w_1 in B-P.
So, we add both the ways.

Vanishing Gradient Problem :-

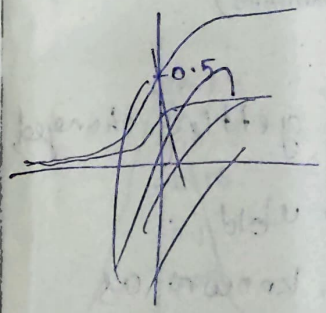


(Mean Squared Error) MSE = Loss = $\frac{1}{2}(y - \hat{y})^2$

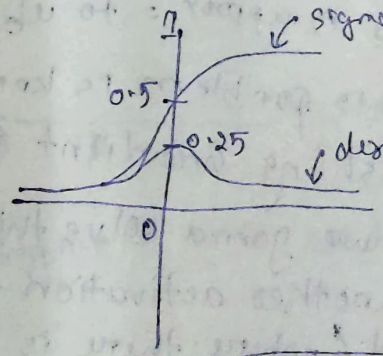
$$w_{1_{new}} = w_{1_{old}} - \eta \frac{dL}{dw_{1_{old}}}$$

$$\frac{dL}{dw_{1_{old}}} = \frac{dL}{dO_{51}} * \frac{dO_{51}}{dO_{41}} * \frac{dO_{41}}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dw_1}$$

At every layer we use,
Sigmoid Activation fun^n $\left[y = \frac{1}{1+e^{-x}}\right]$



O/p → 0 or 1
≥ 0.5 1
< 0.5 0



Derivative = $\boxed{0 \leq \sigma(y) \leq 0.25}$

$a_{31} = \sigma[(O_{41} * w_4) + b]$

↓ sigmoid

0 to 0.25

Let's take an example,

that

$$\frac{dL}{dO_{51}} * \frac{dO_{51}}{dO_{41}} * \frac{dO_{41}}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dw_1}$$

$0.25 * 0.15 * 0.10 * 0.05 * 0.02$

The values are getting ↓ (decreasing)

In short we're going to get a very small value.

small no

$w_{new} = w_{old} - \eta$ (small number)

$w_{new} \approx w_{old}$

Hence w_{new} is hardly getting changed,

w_{new} is approx. = to w_{old}.

So, this problem is known as Vanishing Gradient Problem.

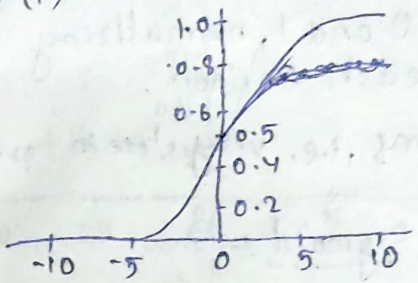
So, how we gonna solve this?

→ Use another activation function instead where there is no vanishing gradient descent.

Activation functions

1) Sigmod function $\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma(x)$

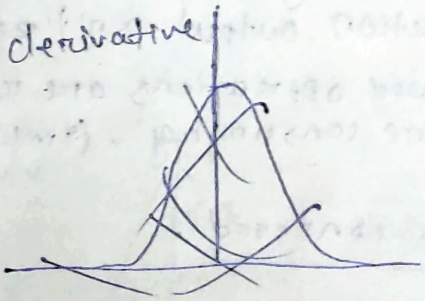


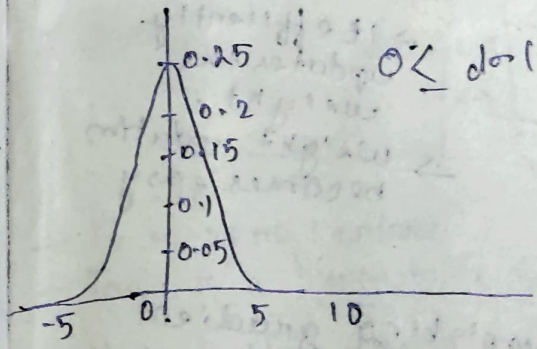
$0.5 \geq = 1$

$0.5 < = 0$

$0 \leq \sigma(x) \leq 1$

$\frac{d\sigma(x)}{dx}$ = derivative





$0 \leq d\sigma(x)/dx \leq 0.25$

O/p of sigmoid = 0 or 1

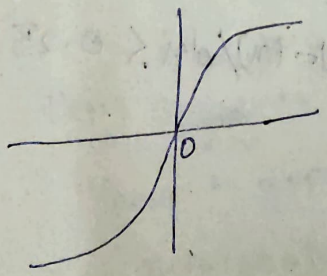
Advantages of Sigmoid :-

1) Smooth gradient, preventing "jumps" in output values
2) O/p bound bet^n 0 and 1, normalizing the output of each neuron.
3) Clear Predictions, i.e. very close to 1 or 0.

Disadvantages of Sigmoid :-

1) Prone to gradient vanishing
2) function output is n't zero centered
3) Power operations are relatively time consuming. (Time complexity ↑)

Zero-centered.



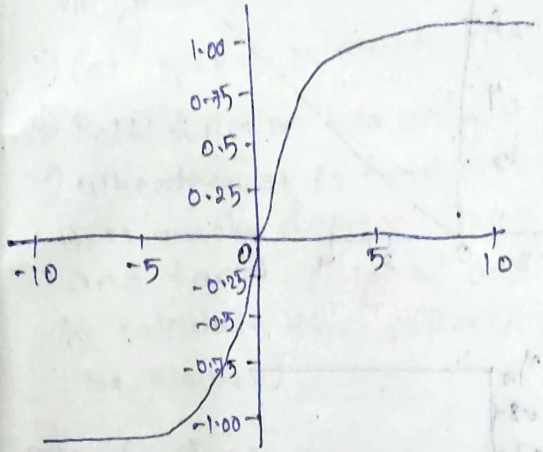
→ It efficiently updates the weight.
→ weight updation becomes easy

If there is a vanishing gradient problem we can't create a deep neural network.

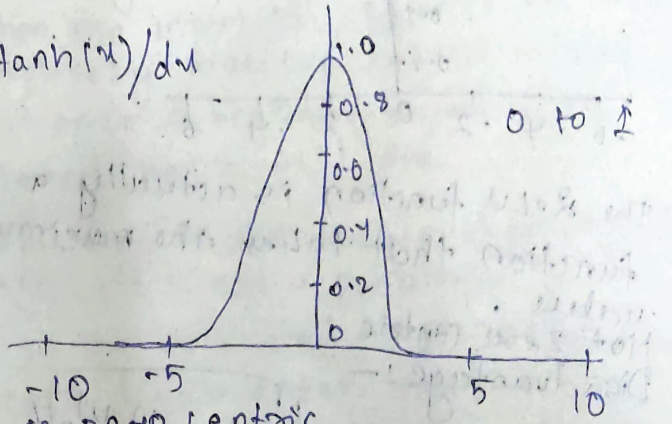
So, New activation fn come up which are :—

2) tanh function

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



-1 to 1

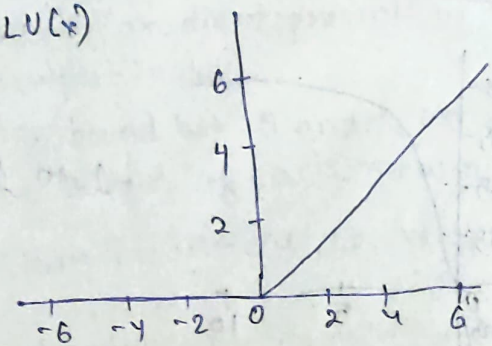


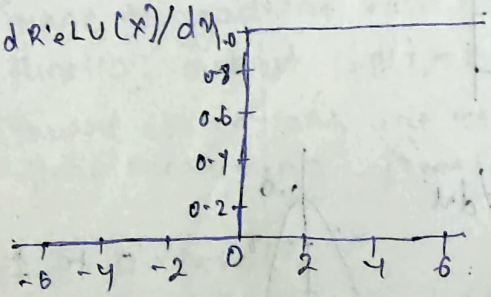
0 to 1

→ It is zero centric
→ Tanh is a hyperbolic tangent function
→ It is better than sigmoid activation function in case of solving vanishing gradient descent.
→ But if we create a very deep neural network. vanishing gradient problem will still exist.

3) ReLU function (Rectified Linear Unit)

$ReLU = max(0, x)$





→ The ReLU function is actually a function that takes the maximum value.

→ Not Zero centric

Disadvantage:-

$\frac{dL}{dw_{old}} = 0$

$w_{new} = w_{old}$

0 or 1 → OK

dead state

Advantages:-

1) when the input is positive, there is no gradient saturation problem.
2) Cal^n speed is much faster
3) ReLU function has only a linear relationship
4) Whether it is forward or backward, it is much faster than sigmoid and tanh. (Sigmod and tanh need to calculate the exponent, which will be slower)

Disadvantages:-

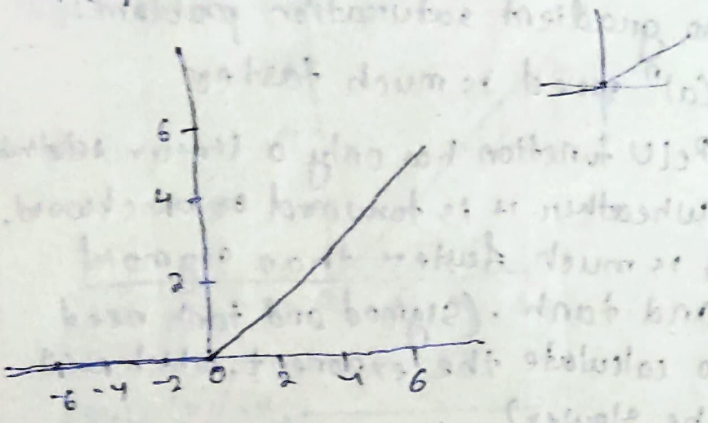
1) when the input is negative, ReLU is completely inactive. which means that once a negative number is entered, ReLU will die.

In this way, in the forward propagation process, it is not a problem. But in the backward propagation process, if you enter ~~enter~~ a negative number, the gradient will be completely zero, which has the same problem as the sigmoid function and tanh function

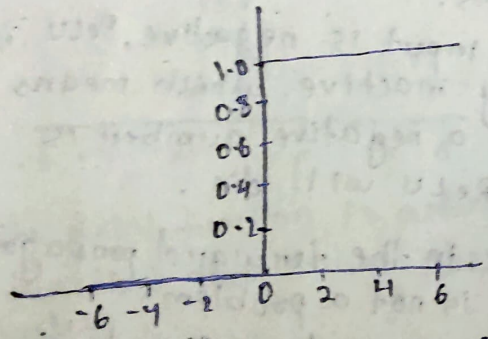
2) We find that the output of the ReLU function is either 0 or a positive number, which means that the ReLU function is not a 0-centric function.

4) Leaky ReLU function (Solve Dead Neuron)

$f(x) = \max(0.01x, x)$



$df(x)/dx$ = Derivative



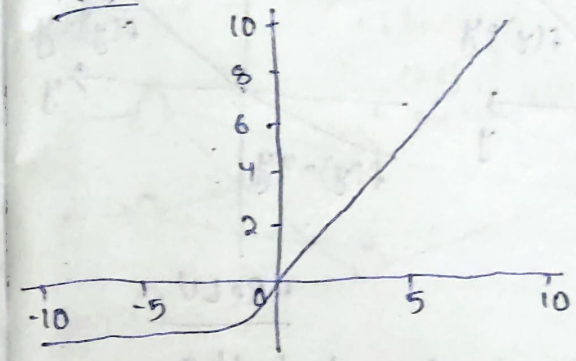
→ In order to solve the Dead ReLU problem, we use Leaky ReLU to set the first half of ReLU 0.01x instead of 0.

→ In theory, Leaky ReLU has all the advantages of ReLU, plus there will be no problems with Dead ReLU, but in actual operation, it hasn't been fully proved that Leaky ReLU is

5) ELU (Exponential Linear Units) function

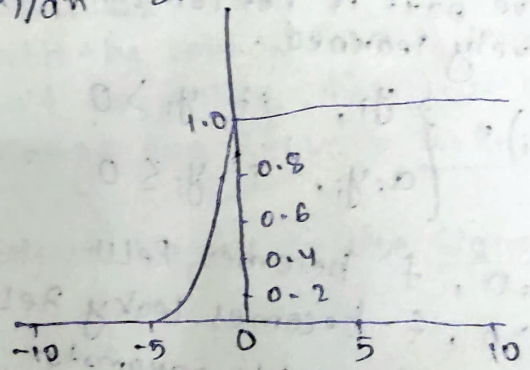
$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

$f(x)$



* there is currently no good evidence in practice that ELU is always better than ReLU

$df(x)/dx$ = Derivative



→ ELU is also proposed to solve the problems of ReLU.

→ ELU has advantages over ReLU

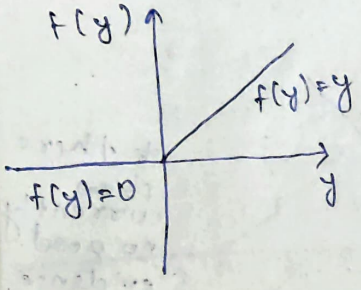
→ No dead ReLU issues

→ the mean of the output is close to 0, zero-centered.

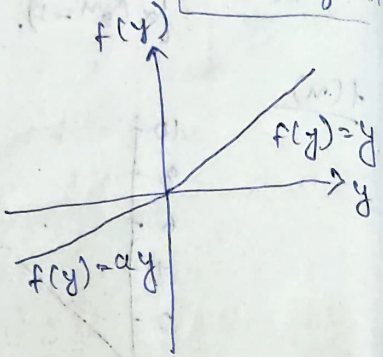
Disadvantage :- It is slightly more computationally intensive. Similar to leaky ReLU, although theoretically better than ReLU *

6) Softmax → Loss fu" → Classification → Categorical Cross Entropy (Multiclass classification)

7) P Relu (Parametric ReLU)



ReLU



PReLU

For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

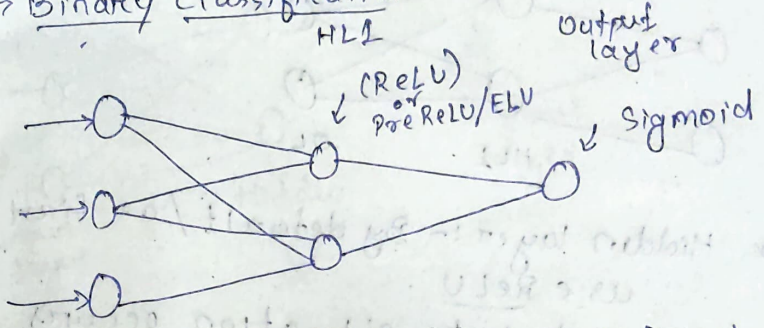
$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

if $a_i = 0$, f becomes ReLU
if $a_i > 0$, f becomes leaky ReLU
if a_i is a learnable parameter, f becomes PReLU

(In learnable parameter it keeps on changing the parameter.)

Technique which Activation fn we should used

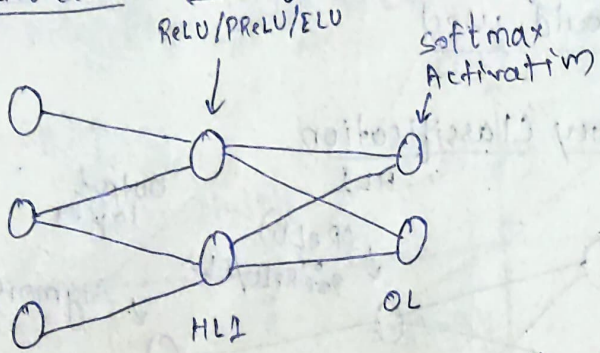
→ Binary Classification



* Hidden layer :- Use ReLU at first.
~~If ReLU isn't~~
If with the help of ReLU if convergence isn't happening, then we can change the ReLU to PreReLU or ELU

* Output Layer :- Use Sigmoid

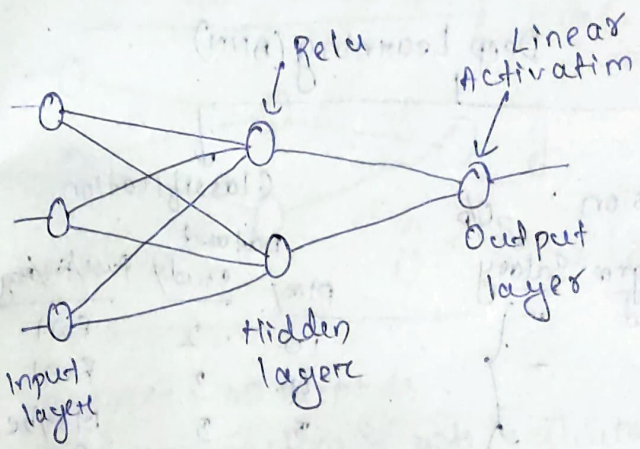
→ Multiclass Classification



* Hidden layer :- By default / at first use ReLU.
If dead state situation occurs then use PreReLU or ELU

* Output layer :- Use softmax.

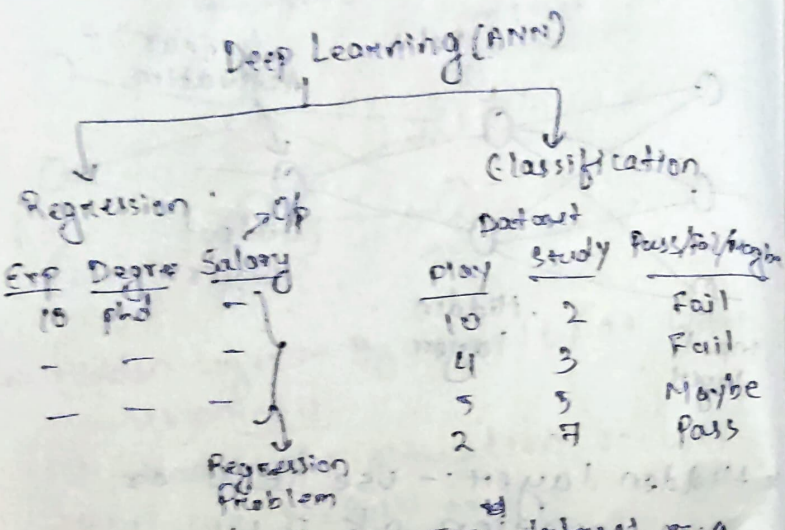
Regression (It means u gonna have continous value)



* Hidden layer :- Use ReLU or any variation of ReLU.

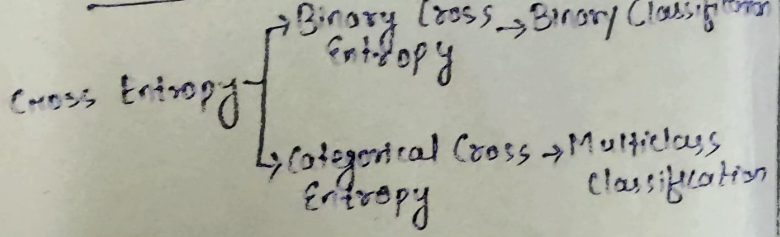
* Output layer :- Linear Activation function and over here there will be a separate loss function

Loss functions

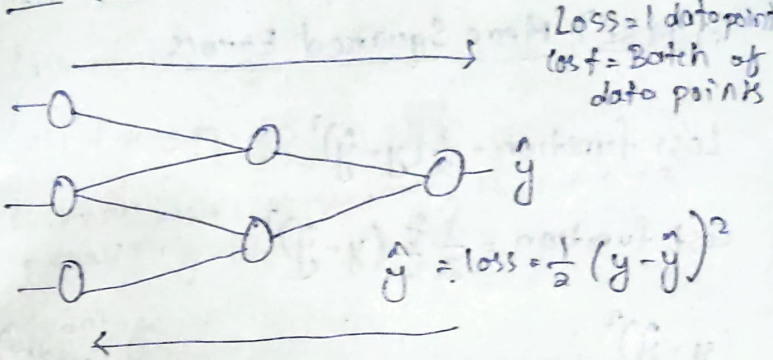


1) Regression
   i) MSE (Mean Squared Error)
   ii) MAE (Mean Absolute Error)
   iii) Huber Loss

2) Classification



Loss function AND cost function



Dataset = 100 records.

→ This loss function is gets calculated for every record.

→ But this is not an efficient scenario Suppose if I've 100 records I can set up a path size. I can basically say at every forward propagation pass 10 records.

when you are passing the 10 records then you calculate the loss for the 10 records.

whenever I pass catch at once. Then the formula get changes instead of loss I will try to write this as cost function.

$$\text{cost} = \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})^2$$

i) Regression

i) (MSE) Mean Squared Error

Loss function = $\frac{1}{2}(y-\hat{y})^2$

Cost function = $\frac{1}{2n}\sum_{i=1}^{n}(y-\hat{y})^2$

$(y-\hat{y})^2$

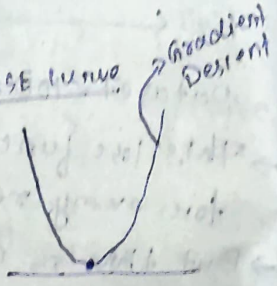
↓

Quadratic Equation

$(a-b)^2 = a^2 + 2ab + b^2$

↓

ax^2+bx+c

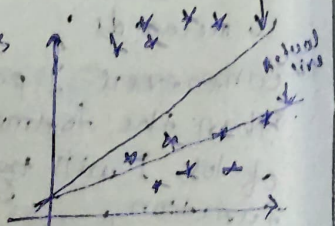


Advantages

1) Differentiable
2) It has only 1 local or global minima.
3) It converges faster

Disadvantage

1) Not Robust to outliers



The reason of shifting the line so much is because we're penalising the error. (squaring the error). Because of that the best fit line will have major shift

ii) (MAE) Mean Absolute Error

Loss function = $\frac{1}{2}|y-\hat{y}|$

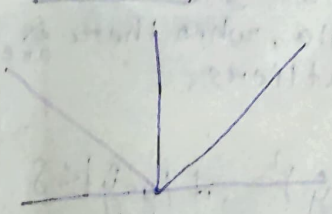
Cost function = $\frac{1}{2}\sum_{i=1}^{n}|y-\hat{y}|$

Here, here we're not squaring it, we're not penalizing it.

Advantage :-

1) Robust to outliers

Here there will be a minor shift not a major one as of MSE.



→ It is time consuming

← Here the derivative will n't be simple to calculate the derivative we've to take subgradient

Subgradient :- We've to divide this line part by part and calculate the slope. (Bez here there is no curve)

iii) Huber Loss

→ It is a combination of MSE and MAE.

$$\text{Loss} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{, if } |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & \text{, otherwise} \end{cases}$$

δ = hyperparameter.

$\boxed{\frac{1}{2}(y-\hat{y})^2}$ => we will use this formula when there is no outliers.

⇒ If the error difference $|y-\hat{y}|$ is less than the hyperparameter.

$\boxed{\delta|y-\hat{y}| - \frac{1}{2}\delta^2}$ => we will use this formula, when there are outliers.

$$\text{cost function} = \frac{1}{2}\sum_{i=1}^{n}(y-\hat{y})^2 \text{ , if } |y-\hat{y}| \le \delta$$

2) Classification

1) Binary Cross Entropy (Binary Classification)

$$\text{Loss} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$

Log loss function.

This is the loss that is used in logistic Regression.

As logistic regression is a binary classification problem.

We can write the above formula as :-

$$\text{Loss} = \begin{cases} -\log(1-\hat{y}) & \text{if } y=0 \\ -\log(\hat{y}) & \text{if } y=1 \end{cases}$$

$\hat{y} = \frac{1}{1+e^{-w}}$ (sigmoid Activation fun in the last layer)

ii) Categorical Cross Entropy (Multiclass Classification)

f_1	f_2	f_3	O/P
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

Good	Bad	Neutral
1	0	0
0	1	0
0	0	1

Loss =

$$L(y_i, \hat{y}_i) = -\sum_{j=1}^{c} y_{ij} * \ln(\hat{y}_{ij})$$

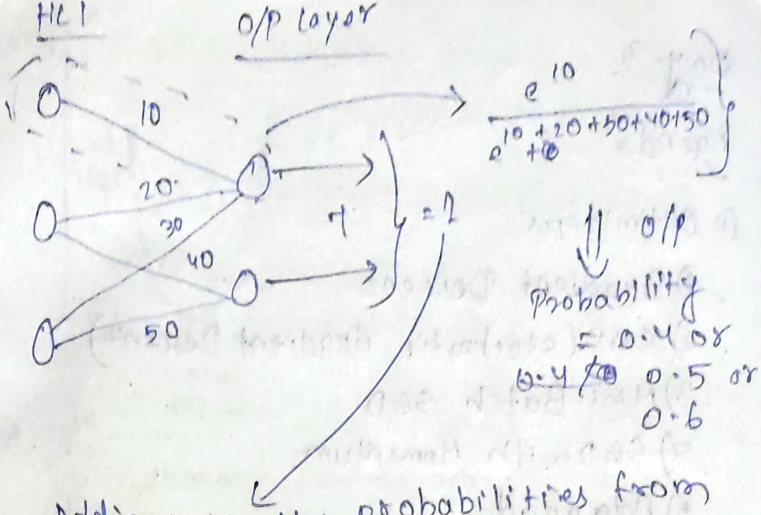
$y_i = [y_{i1}, y_{i2}, y_{i3} \cdots y_{ic}]$ c = no. of categories
i = rows
j = column

$$y_{ij} = \begin{cases} 1 & \text{if the element is in class} \\ 0 & \text{otherwise} \end{cases}$$

$\hat{y}_{ij}$ = Softmax Activation (Applied in the o/p of the multiple classification)

Softmax Activation formula :-

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$



Adding both the probabilities from the output layers will be = 1 ; whichever have higher probability, that class output will be coming up.

* Softmax Activation Function is applied on every output layer for a multiclass classification problem.

ii) Categorical Cross Entropy (Multiclass Classification)

f_1	f_2	f_3	o/p
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

Good	Bad	Neutral
1	0	0
0	1	0
0	0	1

Loss =

$$L(x_i, y_i) = -\sum_{j=1}^{c} y_{ij} * \ln(\hat{y}_{ij})$$

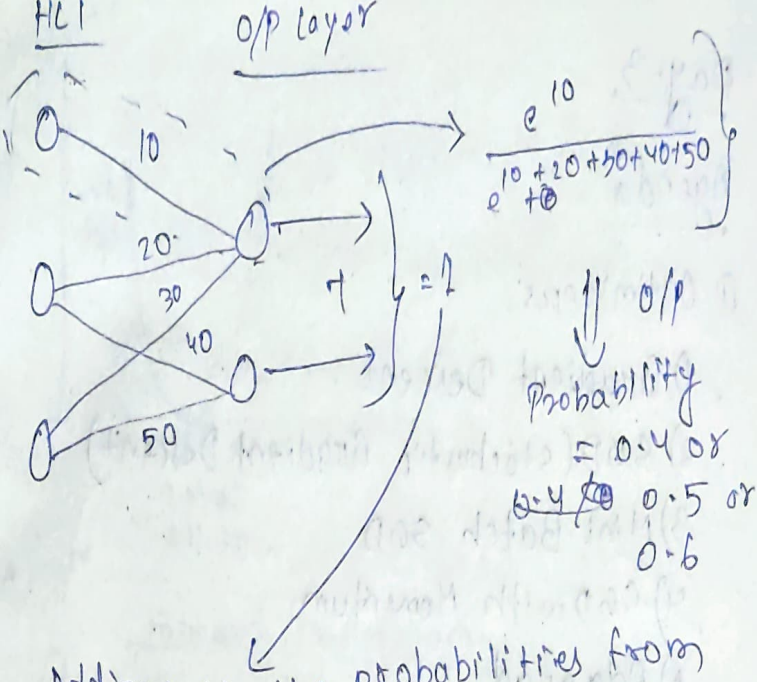
$y_i = [y_{i1}, y_{i2}, y_{i3} \cdots y_{ic}]$ c = no. of category, i = row, j = column

$$y_{ij} = \begin{cases} 1 & \text{if the element is in class} \\ 0 & \text{otherwise} \end{cases}$$

$\hat{y}_{ij}$ = Softmax Activation (Applied in the o/p of the multiple class itself)

Softmax Activation formula :-

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$



Adding both the probabilities from the output layer will be = 1 :

whichever have higher probability, that class output will be coming up.

* Softmax Activation Function is applied on every output layer for a multiclass classification problem.

Day-3

Agenda

① Optimizers
1) Gradient Descent
2) SGD (stochastic Gradient Descent)
3) Mini Batch SGD
4) SGD with Momentum
5) Adagrad
6) RMSPROP
7) Adam Optimizers

② Batch → How many records to be given in a single Epoch
③ Epochs → 1 F.P + 1 B.P = 1 Epoch (Total no. of times the algo will go through the entire dataset)
④ Iterations → Total no. of batches needed to complete 1 epoch

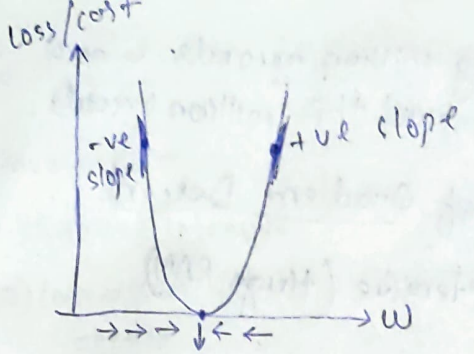
① Optimizers :—

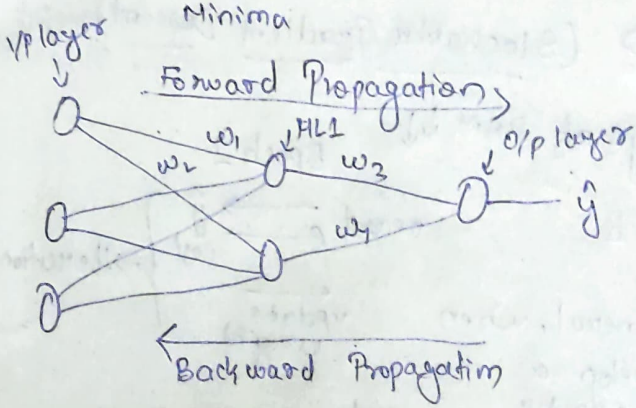
1) Gradient Descent

Weight Updation Formula

$$w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$$

η → learning rate.





Cost function = $\frac{1}{2n} \sum_{i=1}^{n} (y - \hat{y})^2$.

1 Forward Propagation + 1 Backward Propagation = 1 Epoch

If you have million records in the dataset in the F.P at a batch u r taking million record and finding out the cost function so, u r using $\sum_{i=1}^{n}$ → million.

So, if u'r passing million records u need a huge ram to load this million records

Disadvantage of Gradient Descent

1) Resource Extensive (Huge RAM)

2. SGD (Stochastic Gradient Descent)

→ Usage of RAM ↓↓

Epoch 1

1 record → ŷ → cost → update weight → Iteration 1

2 record → ŷ → cost → update weight → Iteration 2

⋮

Million Iterations

→ In general, when we develop a deep learning model, there will be more than 1 epoch.

Suppose let's take 100 epoch

So, 100 million iterations. It will make the process slow.

Disadvantage of SGD

1) Convergence will be very slow.
2) Time complexity will be very high

3. Mini Batch SGD

1000000 → records

batch size = 1000

Total no. of iterations = 1000000/1000 = 1000

Epoch-1

1000 → Iteration 1

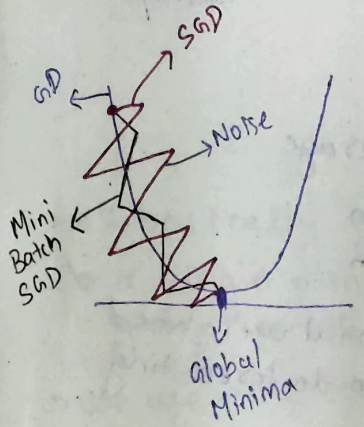
→ Iteration 2

⋮

→ Iteration 1000

Advantage :-

1) Resource Intensive
2) Convergence will be better
3) Time Complexity will improve.



Noise :- The deviation from the path.

SGD > Mini Batch SGD > GD

Q) How do we remove the noise?

→ We use momentum

* Mini batch SGD gives less noise as compared to SGD. But there is noise to remove it we use mini batch SGD with momentum.

Mini batch SGD

SGD with Momentum

Momentum will smoothen the journey and reduce the noise & try to come to the global minima.

Q) How will it smoothen?
→ With the help of exponential moving average

Exponential Weighted / Moving Average

⇓

Time Series

⇓

ARIMA, ARMA (Models where EWA is used)

$$w_{new} = w_{old} - \eta \frac{dL}{dw_{old}}$$

$$w_t = w_{t-1} - \eta \frac{dL}{dw_{t-1}}$$

t = current time stamp

Exponential Weighted Average

t_1 t_2 t_3 t_4 ---- t_n

a_1 a_2 a_3 a_4 ---- a_n

With the help of exponential weighted average we can also do forecasting.

$V_{t_1} = a_1$

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$

β = Hyperparameter

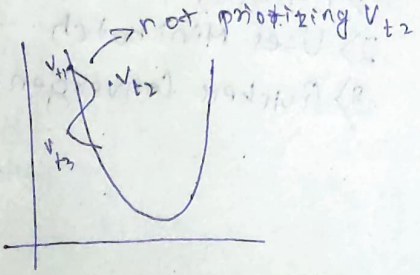
β = It helps to decide which point to prioritize more, prev. one or the present one.

Suppose $\beta = 0.95$

$V_{t_2} = (0.95) * V_{t_1} + (1-0.95) * a_2$

$= 0.95 * V_{t_1} + (0.05) * a_2$

Here, we can see that it is prioritizing the previous point (V_{t_1})



→ It basically gives the control to a single point bet^n the two given points.

→ We use the exponential weighted average in the derivative.

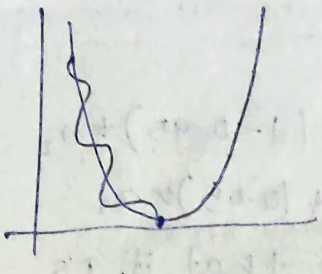
$w_t = w_{t-1} - \eta \frac{dL}{dw_{old}}$

$$\Rightarrow \boxed{w_t = w_{t-1} - \eta \, V_{dw}}$$

$$\boxed{V_{dw_t} = \beta \times V_{dw_{t-1}} + (1-\beta) * \frac{dL}{dw_{t-1}}}$$

⇓

Exponential weighted Avg.



Advantages:-
1) Reduces the Noise
2) Uses Minibatch
3) Quicker Convergence

5) Adagrad (Adaptive Gradient Descent)

Objective:- As we've a learning rate in weight updation. And it is fixed till now, can we do a certain thing that :-

At first, when the point is far away from global minima we can increase the speed by increasing the learning rate.

Later, when the globa point is near the global minima we can decrease the learning rate.

Can we do this?

→ Yes in Adagrad.

$$w_t = w_{t-1} - \eta \frac{dL}{dw_{t-1}} \qquad \eta = \text{fixed}$$

$$\boxed{w_t = w_{t-1} - \eta' \frac{dL}{dw_{t-1}}} \qquad \eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

↓ small no

ϵ is added to avoid divide by zero

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{dL}{dw_t}\right)^2$$

As $\alpha_t \uparrow$ $\eta' \downarrow$ bcz α_t divides η which is fixed $\eta = 0.01$

Bcz of this, we can see that as we go near the global minima, η' decreases.

By this way we're bringing adaptiveness to the learning rate.

η = fixed => adaptive => learning Rate =>

Decreasing => Global Minima.

t=1 η=0.01 ; t=2 η=0.005 ; t=3 η=0.002

→ Here η' will be a very small no. then $w_t \approx w_{t-1}$

So, we've to make sure that α_t shouldn't increase that much.

6 Adadelta And RMSProp

$$\eta' = \frac{\eta}{\sqrt{sdw + \epsilon}}$$

Exponential Weighted Average :-

$sdw = 0$

$$sdw_t = \beta\, sdw_{t-1} + (1-\beta)\left(\frac{dL}{dw_{t-1}}\right)^2$$

Suppose $\beta = 0.95$

$$sdw_t = (0.95)\, sdw_{t-1} + (0.05)\left(\frac{dL}{dw_{t-1}}\right)^2$$

this will not increase that much

$$\Rightarrow \boxed{w_t = w_{t-1} - \eta' \frac{dL}{dw_{t-1}}}$$

$$\Rightarrow \boxed{w_t = w_{t-1} - \frac{\eta}{\sqrt{sdw + \epsilon}} \frac{dL}{dw_{t-1}}}$$

Combining both sdw & Vdw = Adam optimizer

sdw ↓ low learning rate η at a particular point ; Vdw ↓ momentum

7 Adam Optimizer (Best Optimizer)

Momentum (Vdw) + RMSProp (sdw) (Adaptive learning Rate)

↓

Adam Optimizer

$Vdw = 0$ $Vdb = 0$ (↓ bias) $sdw = 0$ $sdb = 0$ (↓ bias)

$$w_t = w_{t-1} - \eta' Vdw$$

$$b_t = w_{b,t-1} - \eta' Vdb$$

$$\eta' = \frac{\eta}{\sqrt{sdw + \epsilon}}$$

$$Vdw_t = \beta \times Vdw_{t-1} + (1-\beta) * \frac{dL}{dw_{t-1}}$$

$$Vdb_t = \beta \times Vdb_{t-1} + (1-\beta) * \frac{dL}{db_{t-1}}$$

ϵ = small no

ϵ is added to avoid divide by zero.

Day-4

Agenda

1) ANN practical implementation
2) Early stopping
3) Black Box Models Vs White Box Models
4) CNN Introduction

Black Box Models Vs White Box Models

Random Forest → Black Box Model

Decision Tree → White Box Model

Linear Reg → White Box Model

ANN → Black Box Model

Xgboost → Black Box Model

Black Box → We can't see how the algo works as it is quite huge and quite big internally.

White Box → We can't observe the internal

Interpretability is possible

Day-5

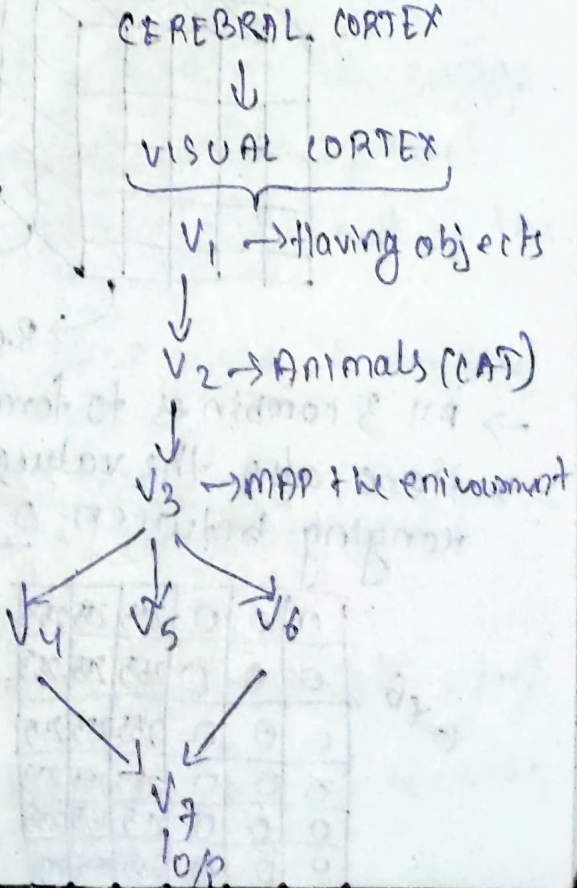
CNN (Convolution Neural Network)

(Image, Video frame)

Agenda

1) CNN Vs Human Brain
2) Convolutional Operation
   - → Convolution
   - → Padding
   - → Stride
   - → Filters (kernels)
3) Max Pooling
4) Flattening
5) Practical Implementation.

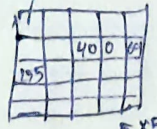
1. CNN vs Human Brain



2 Convolution Operation

Convolution

Images = 0-255 — 'Black & white' & 'RGB'



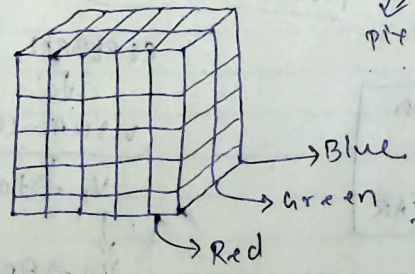
→ We divide the image into some kinds of pixels.

→ Each pixel range = 0-255

let's say → 0 = Black
→ 255 = White

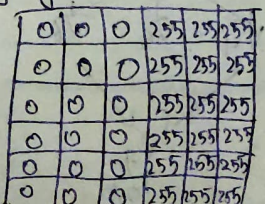
→ Every pixel may have different values based on the image composition.

RGB (coloured Image)



→ All 3 combines to form a image.

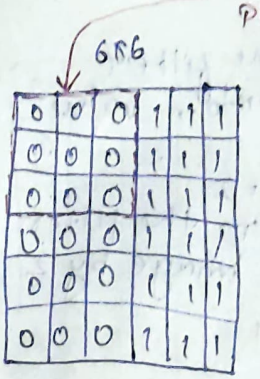
→ Here also the values will be ranging between 0 to 255.

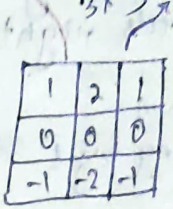


→ We divide each pixel with 255

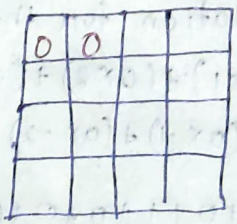
→ To get o/p as 0 or 1

⇒ Step-1 in convolution = bring all the values bet^n 0 — 1.





convolution operation



→ Dividing each pixel with 255 is called as mix-max scaling.

→ The values of the 3x3 kernel/filter is known as horizontal edge filter.

⇒ Step-2:- Placing the filter in the image.

→ Then we perform multiplication of each shell of the image with each shell of the filter.

→ Then add all of it and place it in the O/p (4x4).

→ Calculation for the 1st instance:-

= (0x1)+(0x2)+(0x1)+(0x0)+(0x0)+(0x0)+(0x-1)+(0x-2)+(0x-1)

= 0 (it will be stored in the 1st shell of the 4x4 O/P)

→ Then move/shift the filter one place towards the right.

Stride = Shifting/Moving the filter towards right by a certain value.

Stride = 1
↳ shifting by 1

stride = 2
↳ shifting the filter in the image by 2.

Here, we've taken stride = 1

→ Calculation for the 2nd instance :-

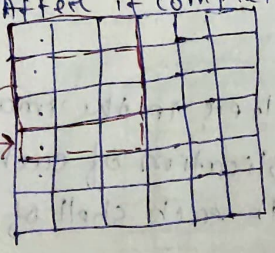
$= (0\times1)+(0\times2)+(1\times1)+(0\times0)+(0\times0)+(0\times0)$
$+(0\times-1)+(0\times-2)+(1\times-1)$

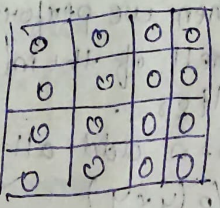
$= 0+0+1+0+0+0+0+0-1$

$= 0$ (It will be stored in the 2nd slot)

→ 3rd & 4th instance = 0

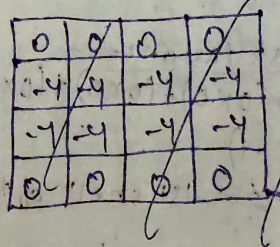
→ After it completes the shifting towards right, shift to down from the starting point of that row. Shift to the down by 1



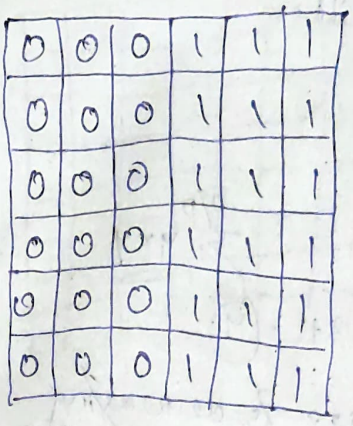


→ So, the final o/p ⇒

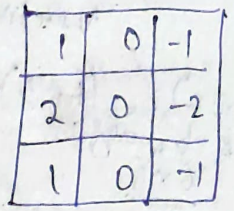
But let's consider the got this :

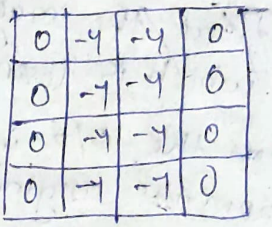


→ The above operation is specifically called as convolution.



(vertical edge filter)





Applying min-max scaling

Min-Max Scaling :-

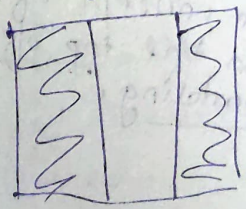
Smaller value → 0

higher value = 255 will become

smaller value
$= -4 \rightarrow 0$

higher value
$= 0 \rightarrow 255$

0 → black colour
255 → white colour



Vertical edge

Filter → Round shape filter
horizontal
Vertical

(n=6) size of image → 6×6
(f=3) → filter → 3×3
O/P → 4×4

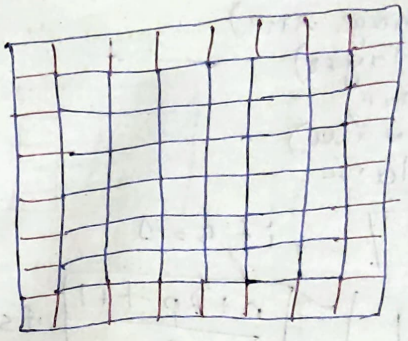
$n-f+1 = 6-3+1 = 4$

there the image size is decreasing which is losing some information.

→ there the O/P image size is decreasing from the orginal I/P image.
→ which means losing some information.
→ To avoid this, we can come up with something called padding.

→ If you've a image of 6×6 & wants an O/P of image of 6×6.
Then apply padding on the orginal I/P image.

→ The concept of applying a layere on the top of the image is called padding.



After applying padding on a 6×6 image.

It will become a 8×8 image.
(But the actual image will be only 6×6)
There will be an external layer added to it.

Types of padding:-
1) → zero padding
2) whatever the nearest value to the padding, put (insert) that value into padding.

Before Padding :-
n=6, f=3
$n-f+1$
→ $6-3+1 \Rightarrow 4 = O/P$
there, image size is decreasing.
Information Loss. takes place.

After padding :-
n=8, f=3
$n-f+1$
→ $8-3+1 \Rightarrow 6 = O/P$
there, actual image size is maintained
No. information is lossed.

n = 6 (Actual image size)
p = 1 (padding layer)
f = 3 (filter size)
s = 1 (stride value)

Updated formula:-

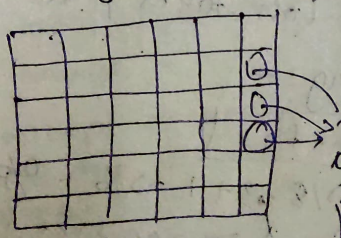
$\frac{n+2p-f+1}{}$; if $s = 1$

$\frac{n+2p-f+1}{s}$; if $s > 1$

$= 6 + 2(1) - 3 + 1$
$= 6 + 2 - 3 + 1$
$= 6$

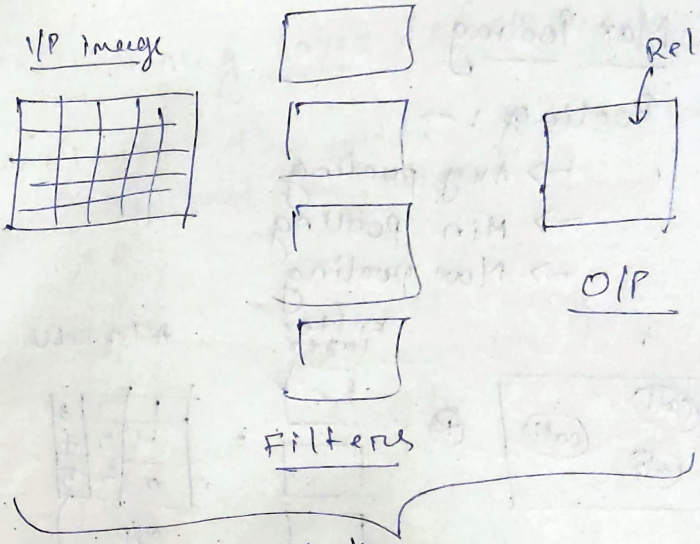
→ As in ANN during the back propagation we need to update the weights.
→ Like that in CNN we need to update the filters based on the input image.
→ Initially we may initialize some filter randomly, but later we need to update the filters

After convolution takes place, we get a 6×6 image, then apply relu & activation function on the value of each shell.



Apply relu on every shell.

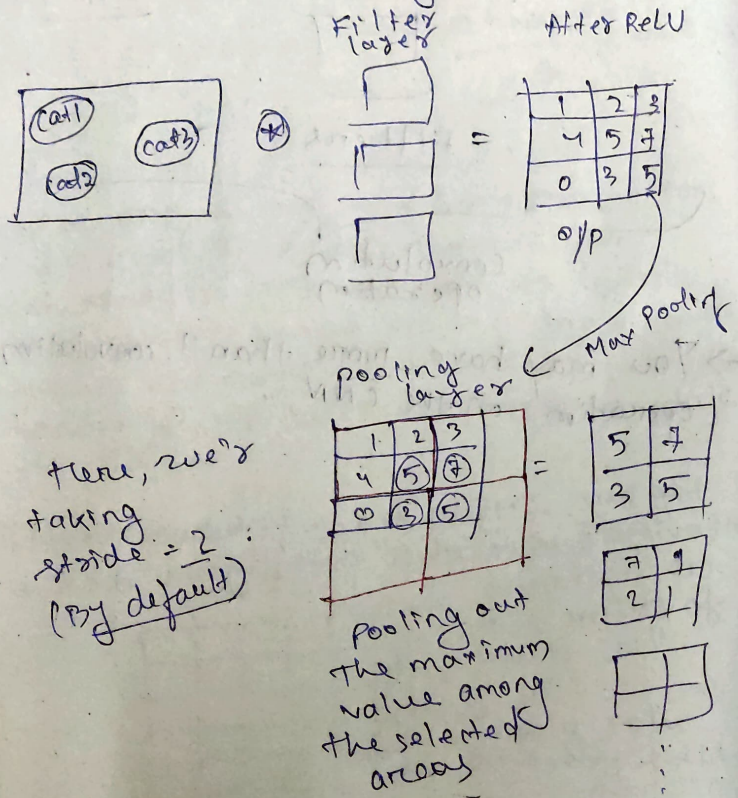
Relu = max(0, x)



→ You may have more than 1 convolution operation on the CNN.

3 Max Pooling

Pooling :-
- → Avg pooling
- → Min pooling
- → Max pooling



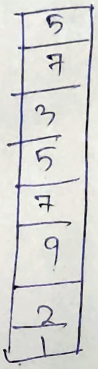
Here, we're taking stride = 2 (By default)

Avg Pooling → In avg pooling, find out the avg value from the selected.

Min Pooling → Find out the minimum value among the selected.

4 Flattening Layer / ANN Dense layer

→ Flatten the entire Max pooling o/p. (Just elongating it)



Just Attaching downwards in a single column.

Overall View